Clasificación de Fashion-MNIST con una Red Neuronal Feedforward Multicapa implementada en PyTorch

Martina L. Cesca*
Departamento de Ciencias e Ingeniería de la Computación (DCIC), Universidad Nacional del Sur, 8000 Bahía Blanca, Buenos Aires, Argentina

Matías G. García Casas**
Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

María Victoria Mascaró de Gárate***
Facultad de Ciencias Económicas (FCE), Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina
(Dated: 26 de noviembre de 2025)

Se entrenó una red neuronal feedforward multicapa en PyTorch para resolver un problema de clasificación multiclase sobre imágenes del dataset Fashion-MNIST. Se analizó el impacto del optimizador, la tasa de aprendizaje y la arquitectura en la convergencia y precisión del modelo.

I. INTRODUCCIÓN

Las redes neuronales artificiales (RNAs) [1] son modelos computacionales compuestos por unidades interconectadas que procesan información de manera distribuida. Entre ellas, las **redes neuronales feedforward multicapa** (*Multilayer Perceptrons*, MLP) [2] representan una de las arquitecturas más utilizadas para el aprendizaje supervisado. Estas redes propagan la información hacia adelante a través de capas sucesivas —de entrada, ocultas y salida— ajustando sus pesos mediante retropropagación del error.

En problemas de **clasificación multiclase**, las MLP aprenden a asociar patrones de entrada con categorías discretas mediante la minimización de una función de pérdida. Su capacidad para modelar relaciones no lineales las hace adecuadas para tareas de reconocimiento de imágenes, texto o señales.

El presente trabajo tiene como objetivo entrenar una red neuronal *feedforward* MLP en *PyTorch* para resolver un problema de *clasificación multiclase* sobre imágenes del dataset *Fashion-MNIST*, analizando el impacto de distintos hiperparámetros (optimizador, tasa de aprendizaje y arquitectura) en su desempeño.

II. TEORÍA

El conjunto de datos contiene 60.000 imágenes de entrenamiento y 10.000 de prueba, correspondientes a prendas de vestir en escala de grises (1 canal, 28×28 píxeles). Cada imagen pertenece a una de diez clases balanceadas: remera, pantalón, buzo, vestido, abrigo, sandalia, camisa, zapatilla, bolso o bota. Los valores de los píxeles se normalizaron con media 0.5 y desviación estándar 0.5, reescalando el rango a $[-1, 1]$ para estabilizar el descenso por gradiente, evitar gradientes excesivamente grandes o pequeños y acelerar la convergencia del entrenamiento.

En redes neuronales, el proceso de entrenamiento consiste en ajustar progresivamente los pesos para minimizar una función de pérdida. La pérdida utilizada en problemas de clasificación multiclase es la **Cross Entropy Loss**, que cuantifica la diferencia entre la distribución predicha y la real, penalizando con mayor intensidad las predicciones incorrectas con alta confianza. El **optimizador** controla cómo se actualizan los pesos a partir del gradiente. El **learning rate** determina el tamaño del paso que da el optimizador: valores muy altos pueden provocar oscilaciones y falta de convergencia, mientras que valores demasiado bajos generan un aprendizaje lento. El **batch size** define cuántas muestras se procesan antes de actualizar los pesos; tamaños pequeños introducen más ruido en el gradiente y favorecen la generalización, mientras que tamaños grandes hacen el aprendizaje más estable, aunque con mayor riesgo de sobreajuste. Las **epochs** indican cuántas veces el modelo recorre el conjunto completo de entrenamiento.

Un elemento fundamental en una red neuronal es la **función de activación**, que define cómo cada neurona transforma la información que recibe antes de enviarla a la siguiente capa. Estas funciones introducen no linealidad en el modelo, permitiendo que la red aprenda relaciones complejas; sin ellas, incluso una red con muchas capas sería equivalente a una regresión lineal. Históricamente se utilizaron funciones como la sigmoidea o la tangente hiperbólica, pero ambas presentan problemas de saturación: cuando la entrada toma valores extremos, sus derivadas se vuelven casi cero, impidiendo que los gradientes fluyan durante el entrenamiento (fenómeno conocido como **vanishing gradient**). En contraste, la función **ReLU (Rectified Linear Unit)** se ha convertido en el estándar en redes profundas porque es computacionalmente eficiente y evita la saturación en la mayor parte del rango de entrada, acelera la convergencia y mejora el

desempeño.

El diseño de una red neuronal implica definir cuántas **capas ocultas** tendrá y cuántas **neuronas** conformarán cada una de ellas. Tanto la cantidad de capas como el número de neuronas determinan la capacidad del modelo para aprender patrones complejos. Si la red tiene pocas capas o muy pocas neuronas, puede no capturar la estructura del problema, resultando en *underfitting*. En cambio, si se agregan demasiadas capas o un número excesivo de neuronas, la red puede terminar memorizando los datos de entrenamiento, causando *overfitting* y aumentando innecesariamente el costo computacional. Diseñar la arquitectura implica encontrar un equilibrio entre capacidad de representación, velocidad de entrenamiento y generalización.

El método **dropout** se utiliza para reducir el sobreajuste dentro de la red neuronal. Consiste en apagar aleatoriamente un porcentaje de neuronas durante el entrenamiento, impidiendo que el modelo dependa de rutas específicas o patrones demasiado rígidos. De esta forma, la red aprende representaciones más robustas y distribuidas, mejorando su desempeño en datos no vistos. No obstante, un valor de dropout excesivo puede eliminar demasiada información y dificultar el aprendizaje, por lo que su selección también requiere equilibrio.

Para entrenar y evaluar el modelo, el conjunto de datos se dividió en dos partes: **entrenamiento** y **validación**. El primero se utiliza para ajustar los parámetros del modelo, mientras que el segundo permite estimar su capacidad de generalización y detectar posibles signos de **sobreajuste**. Durante el proceso se monitorean métricas como la pérdida y la precisión en validación, lo que facilita identificar si el modelo está aprendiendo de forma efectiva o simplemente memorizando los datos.

El punto de partida fue un modelo **baseline** constituido por una red neuronal (*fully-connected*) con una capa de entrada, dos capas ocultas de **128 y 64 neuronas** respectivamente y una capa de salida con **10 neuronas**, una por cada clase a predecir. En cada capa oculta se utilizó la función de activación **ReLU** y se aplicó **dropout del 20 %**. El entrenamiento se realizó durante **30 epochs**, con **batch size = 100** y función de pérdida **Cross-Entropy**, utilizando el optimizador **Stochastic Gradient Descent (SGD)** y tres valores de tasa de aprendizaje: 10^{-1}, 10^{-2} y 10^{-3}. El objetivo fue analizar la sensibilidad del modelo ante diferentes magnitudes del paso de actualización.

En el **Experimento 2** se repitió la misma arquitectura y configuración general, reemplazando el optimizador por **Adam**, que ajusta de forma adaptativa la tasa de aprendizaje efectiva de cada parámetro Se probaron tres valores de *learning rate* (10^{-2}, 10^{-3}, 10^{-4}), buscando determinar si la adaptación dinámica de Adam mejoraba la convergencia y estabilidad frente a SGD.

Es importante notar que, en la práctica, los valores numéricos de la tasa de aprendizaje no son directamente comparables entre distintos optimizadores. Por ejemplo, un *learning rate* de 10^{-3} en Adam suele producir un comportamiento similar al de 10^{-2} en SGD, lo cual refleja que los valores numéricos no son directamente comparables entre optimizadores. Esto no implica que Adam utilice pasos más pequeños, sino que su mecanismo de ajuste adaptativo escala internamente las actualizaciones de cada parámetro. En otras palabras, aunque el valor numérico sea menor, el efecto efectivo sobre los pesos puede ser equivalente o incluso mayor. Por ello, Adam tiende a funcionar adecuadamente con tasas de aprendizaje *aparentemente más bajas* que las empleadas en SGD.

En el **Experimento 3** se amplió la arquitectura a **tres capas ocultas** de **256, 128 y 64 neuronas** respectivamente, manteniendo activación ReLU y dropout del 20 %. Se entrenó con **Adam** y los mismos valores de *learning rate* que en el experimento anterior. Este análisis permitió evaluar el impacto de una mayor profundidad sobre la capacidad de generalización y el riesgo de *overfitting*.

En el **Experimento 4** se implementó una arquitectura **convolucional inspirada en LeNet-5**, compuesta por dos capas convolucionales (6 y 16 filtros de tamaño 5×5), seguidas de capas *max-pooling*, y tres capas totalmente conectadas de **120, 84 y 10 neuronas**. Este modelo fue entrenado con **Adam** y los mismos *learning rates* que en los dos experimentos anteriores, buscando aprovechar la capacidad de las CNN para capturar patrones espaciales en las imágenes y comparar su desempeño frente a las redes densas tradicionales.

III. RESULTADOS

A continuación se presentan los resultados obtenidos para los distintos experimentos, analizando la evolución de la pérdida (*Loss*) y la exactitud (*Accuracy*) en entrenamiento y validación. Se parte de un modelo **baseline** y se introducen modificaciones controladas para evaluar el efecto del optimizador, la tasa de aprendizaje y la arquitectura.

La Tabla I resume las configuraciones y la Figura 1 muestra las curvas correspondientes.

En el **Experimento 1 (SGD)**, el modelo mostró una alta sensibilidad al valor del *learning rate*. Con $LR = 10^{-1}$, la pérdida mostró oscilaciones y el modelo no logró una convergencia estable. Con $LR = 10^{-2}$, se alcanzó un buen equilibrio entre velocidad y estabilidad (loss $\approx$ 0,3, accuracy $\approx$ 0,88). Finalmente, con $LR = 10^{-3}$, el aprendizaje se volvió muy lento.

El **Experimento 2 (Adam)** mejoró la estabilidad y velocidad de convergencia respecto de SGD. Con $LR = 10^{-2}$ el entrenamiento mostró volatilidad. Con $LR = 10^{-3}$, la pérdida de validación se estabilizó cerca de 0.35 desde las primeras épocas, mientras que la de entrenamiento continuó descendiendo por debajo de 0.2, indicando leve sobreajuste. Por último, con $LR = 10^{-4}$, el

Experimento nº	n_1	n_2	n_3	capas ocultas	tipo	Optimizador	learning-rate	Objetivo
1	128	64	-	2	feedforward	SGD	$10^{-1}, 10^{-2}, 10^{-3}$	Experimento base, según el learning-rate.
2	128	64	-	2	feedforward	Adam	$10^{-2}, 10^{-3}, 10^{-4}$	Cambiar el optimizador, según el LR.
3	256	128	64	3	feedforward	Adam	$10^{-2}, 10^{-3}, 10^{-4}$	Aumentar capas y neuronas, según LR.
4	-	-	-	4	CNN	Adam	$10^{-2}, 10^{-3}, 10^{-4}$	Cambiar el tipo de RNA, según LR.

Tabla I: Lista de experimentos en donde se varían distintos hiperparámetros de un experimento base. En todos los casos, los modelos se entrenaron durante 30 épocas, con un tamaño de lote de 100 y dropout de 0.2.

comportamiento fue más suave y estable, alcanzando en validación una pérdida cercana a 0.3 y una exactitud próxima al 90 % entre las épocas 15 y 20. En conjunto, Adam mostró una convergencia más rápida y una mejor capacidad de optimización que SGD. El desempeño óptimo se alcanzó con un learning rate moderado o bajo, lo que confirma que su ajuste sigue siendo importante para la estabilidad del entrenamiento.

En el **Experimento 3 (Aumento de complejidad del modelo)**, el modelo logró resultados similares al experimento anterior, con pérdida cercana a 0.3 y exactitud aproximándose al 90 %. Sin embargo, se observó un leve sobreajuste hacia las últimas épocas, evidenciado por una brecha creciente entre las curvas de entrenamiento y validación. Estos resultados confirman que incrementar la complejidad no garantiza mejoras sustanciales y resaltan la importancia de balancear capacidad de representación y regularización. El criterio de parsimonia sugiere mantener una arquitectura más simple, dado que la complejidad adicional no aportó mejoras sustanciales.

En el **Experimento 4 (Red Convolucional)**, para un $LR = 10^{-2}$, tanto la pérdida como la exactitud mostraron gran inestabilidad. Con $LR = 10^{-3}$, la pérdida de validación se estabilizó cerca de 0.3 hacia la época 6, alcanzando una exactitud cercana al 90 %, superior a las redes totalmente conectadas. A partir de la época 20 se observó un leve sobreajuste; mientras que con $LR = 10^{-4}$ el aprendizaje fue más progresivo aunque sin meseta clara.

En términos generales, la **CNN** logró el mejor equilibrio entre desempeño y generalización, y el optimizador **Adam** con $LR = 10^{-3}$ ofreció la combinación más estable y eficiente entre rapidez y precisión.

IV. DISCUSIÓN Y CONCLUSIONES

A partir de los experimentos realizados, se observa que el modelo *baseline* logró clasificar las imágenes con un desempeño inicial aceptable, mostrando una disminución progresiva de la pérdida y un incremento sostenido en la precisión a lo largo de las épocas.

El optimizador Adam mostró una convergencia más rápida y estable que SGD, validando su eficacia para este tipo de tareas.

La variación del *learning rate*, confirmó los patrones esperados: valores altos aceleran el aprendizaje pero pueden provocar oscilaciones e inestabilidad, mientras que valores bajos generan curvas más suaves, aunque con convergencia más lenta.

El aumento de la complejidad de la red mejoró la capacidad de representación y permitió una reducción adicional del error, pero también incrementó el riesgo de *overfitting*.

Finalmente, la implementación de una red convolucional resultó ser la más adecuada para el problema, dado que este tipo de arquitectura está específicamente diseñada para capturar patrones espaciales y de textura presentes en imágenes. La CNN alcanzó los mejores resultados tanto en precisión de validación como en estabilidad del entrenamiento, confirmando que las arquitecturas convolucionales se adaptan mejor a la naturaleza visual del dataset *Fashion-MNIST*.

Aunque el *dropout* mitigó el sobreajuste, podrían incorporarse estrategias adicionales como *batch normalization*, *early stopping* o *data augmentation*. Como trabajo futuro, se propone explorar el uso del dataset en color (RGB) para analizar si la información cromática mejora el rendimiento, así como evaluar arquitecturas más profundas o convolucionales.

* marticesca25@gmail.com
** matias.garcia.casas.929@mi.unc.edu.ar
*** mvictoriamascaro@gmail.com

[1] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[2] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, D.C., 1962.

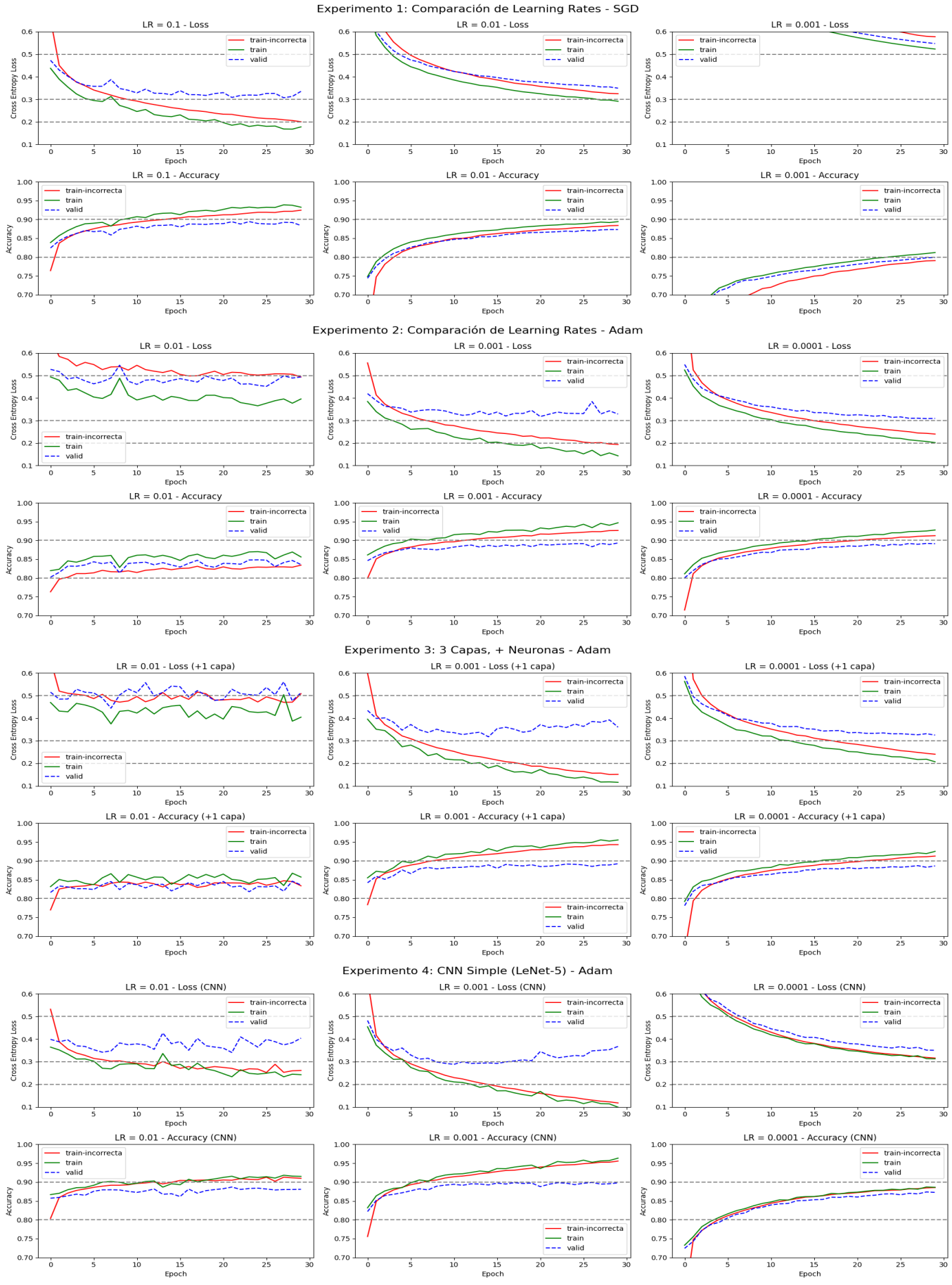

Figura 1: Evolución de la pérdida (Cross Entropy Loss) y la precisión (Accuracy) en 4 experimentos. Se comparan 3 curvas: entrenamiento activo (rojo), entrenamiento sin actualización de pesos (verde) y validación (azul discontinua).